KOMPILÁTORY: Syntaktická analýza (Úvod, metódy zhora nadol)

Jana Dvořáková
dvorakova@dcs.fmph.uniba.sk

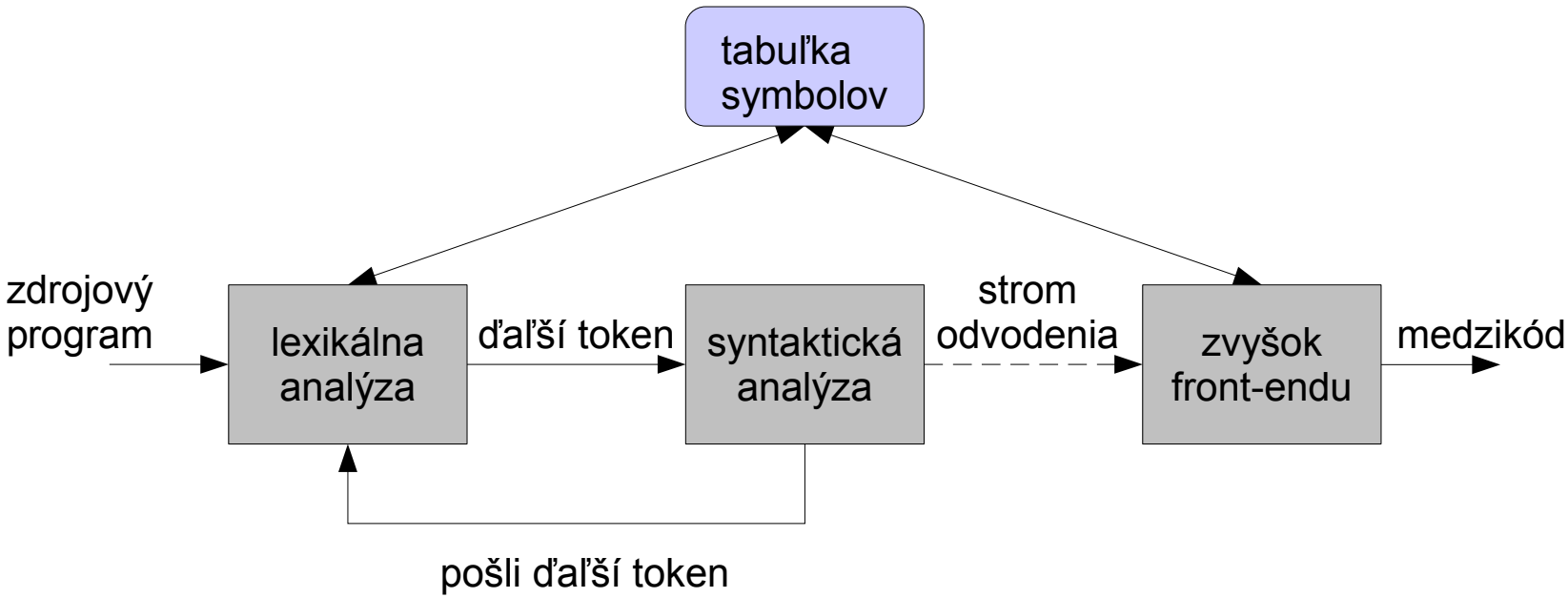
Syntaktická analýza: Úvod

Úlohy syntaktickej analýzy

1. Číta postupnosť tokenov generovanú lexikálnym analyzátorom a konštruuje strom odvodenia/syntaktický strom
   - Overuje, či vstupný reťazec môže byť generovaný CF gramatikou pre vstupný jazyk
2. Hlási syntaktické chyby ”inteligentným” spôsobom
   - Vie sa zotaviť z bežných chýb tak, aby sa mohlo pokračovať v spracovávaní vstupu

- Pracuje na úrovni *bezkontextových jazykov*

Rozhrania parsera



Metódy syntaktickej analýzy

1. Univerzálne metódy
   - Cocke-Younger-Kasami, Earley
   - Všetky CF gramatiky, ale neefektívne ($O(n^3)$)
2. Metódy zhora-nadol
   - Konštruujú strom odvodenia od koreňa k listom v preorderi (ľavé krajné odvodenie)
   - LL gramatiky
   - Aj ručne implementované parsery
3. Metódy zdola-nahor
   - Konštruujú strom odvodenia od listov ku koreňu (pravé krajné odvodenie)
   - LR gramatiky
   - Zvyčajne parsery skonštruované automatickými nástrojmi

Ošetrenie chýb

- Veľká časť chýb sa detekuje práve počas syntaktickej analýzy
  - Množstvo chýb je syntaktickej povahy
  - Chyby sú odhalené, keď postupnosť tokenov nezodpovedá pravidlám gramatiky
  - Moderné parsovacie metódy sú schopné detekovať chyby veľmi efektívne
- Ciele:
  - Jasné a presné hlásenie chýb prítomných v programe
  - Rýchle zotavenie z každej nájdenej chyby
  - Spracovanie korektných programov by sa nemalo výrazne spomaliť
- LL, LR metódy detekujú chyby hneď, ako je to možné
  - Majú vlastnosť *životaschopného prefixu* a chyba sa odhalí hneď ako prefix vstupu nie je prefixom žiadneho slova

Ošetrenie chýb

- Hlásenie chyby
  1. Miesto detekovania chyby
     - Je pravdepodobné, že miesto skutočnej pozície chyby je blízko
     - Často sa vypíše celý riadok so smerníkom na presnú pozíciu
  2. Diagnostická správa
     - Napr. "missing semicolon at position"
- Zotavenie sa z chyby
  - Existuje množstvo stratégií
  - Parser sa pokúša dostať do stavu, z ktorého môže pokračovať v spracovávaní vstupu
  - Problém: zotavenie z jednej chyby spôsobí ďaľšie, riešením môže byť *konzervatívna stratégia*
- Je potrebný kompromis - brať do úvahy iba chyby, ktoré sú pravdepodobné a má zmysel ich spracovať

Stratégie zotavenia sa z chýb

1. Zotavenie v móde paniky
   - Zvolí sa množina *synchronizačných tokenov* (vzhľadom na vsupný jazyk)
     - Zvyčajne oddeľovače: **;**, **end**,...
   - Pri detekovaní chyby sa vynechá časť vstupu až kým sa nenájde synchronizačný token

   (+) Metóda jednoduchá na implementáciu, nikdy sa nezacyklí

   (−) Vynechá sa značná časť vstupu a v nej sa už nekontrolujú chyby
2. Zotavenie na úrovni frázy
   - Keď sa nájde chyba, parser vykoná lokálnu korekciu
     - Prefix neprečítaného vstupu sa nahradí iným reťazcom

   (−) Je zložité ošetrovať chyby, ktoré sa vyskytujú ďaleko pred bodom detekcie

Stratégie zotavenia sa z chýb

3. Zotavenie pomocou chybových pravidiel
   - Rozšírime gramatiku o chybové pravidlá
     - Pravidlá pre generovanie chybných konštrukcií
   - Z gramatiky skonštruujeme parser, ktorý pri použití chybového pravidla generuje diagnostické hlásenie
   - Musíme dopredu vedieť, ktoré chyby chceme ošetrovať
4. Zotavenie pomocou globálnych korekcií
   - Používajú sa algoritmy pre hľadanie najvýhodnejšej korekcie (vyžadujúcej minimálnu postupnosť zmien)
   - "Drahé"na implementáciu vzhľadom na priestor aj čas
   - Zaujímavé z teoretického hľadiska

Bezkontextová gramatika

$G = (N, T, P, S)$

- T - konečná množina terminálov, z pohľadu synt. analýzy množina tokenov posielaných lex. analyzátorom
  - **id**, **:=**, **uint**, ...
- N - konečná množina neterminálov, syntaktické konštrukcie
  - *blok*, *príkaz*, *výraz*, ..
- P - konečná množina pravidiel $N \rightarrow (N \cup T)^*$

Odvodenie

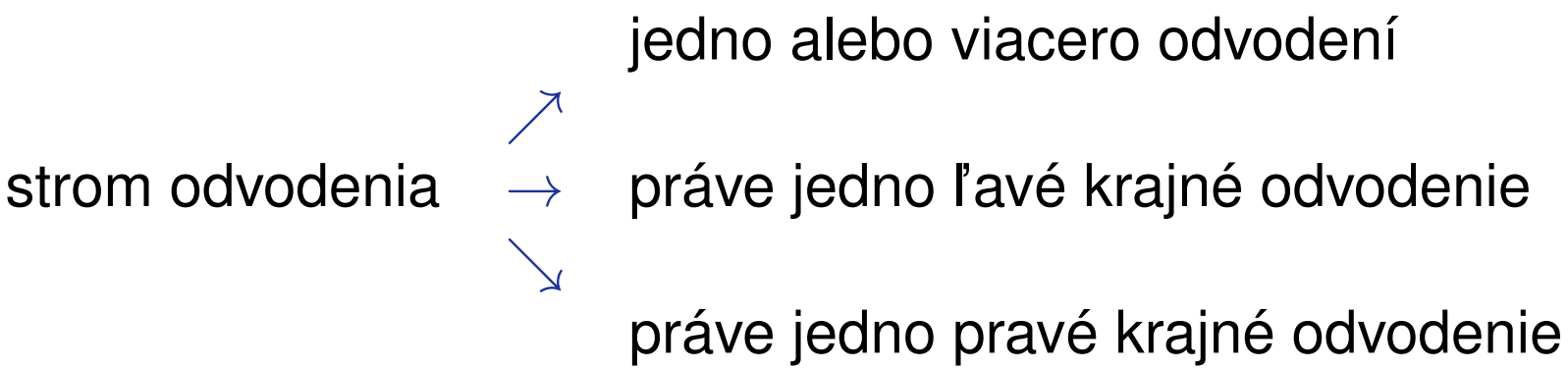
- *Krok odvodenia:*

$$\alpha A \beta \Rightarrow \alpha \gamma \beta, \text{ kde } \alpha, \gamma \in (N \cup T)^*, A \in N, A \to \gamma \in P$$

- *Odvodenie:* $S \Rightarrow^* \alpha$
  - Reťazec $\alpha \in (N \cup T)^*$ je *vetná forma* gramatiky G, ak sa navyše skladá iba z terminálov, je to *slovo* generované G
- Pri každom kroku odvodenia máme dva výbery
  1. Výber neterminálu, ktorý prepisujeme
  2. Výber pravidla pre daný neterminál, ktoré použijeme
- Pri parsovaní má význam ľavé a pravé krajné odvodenia
  - Vždy prepisujeme najľavejší resp. najpravejší neterminál

Strom dvodenia

- Grafická reprezentácia odvodenia, každý vnútorný uzol zodpovedá aplikácii pravidla
- Neznázorňuje poradie aplikovania pravidiel
- Odvodenie zodpovedá konštrukcii stromu odvodenia zhora nadol



Úpravy gramatiky

1. Eliminácia *nejednoznačnosti*
   ($\rightarrow$ parsovanie zhora-nadol aj zdola-nahor)
   - Nejednoznačná gramatika: Pre nejaké slovo existujú dva rôzne stromy odvodenia
   - Konflikty pri parsovaní, rieši sa napríklad ručným pridaním pravidiel pre ich riešenie
   - V niektorých prípadoch môže byť nejednoznačnosť eliminovaná úpravou gramatiky

Úpravy gramatiky

2. Eliminácia *ľavej rekurzie* ($\rightarrow$ parsovanie zhora-nadol)

   - Máme $A \Rightarrow^{+} A\alpha$ pre nejaké $A \in N$
   - Pri metódach pracujúcich zhora nadol ľavorekurzívne gramatiky nevieme spracovať

ALGORITMUS 1: Eliminácia priamej ľavej rekurzie

Ak máme A-pravidlá $A \rightarrow A\alpha_1 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \ldots \mid \beta_n$, nahradíme ich

$$\begin{array}{lcl} A & \rightarrow & \beta_1 A' \mid \ldots \mid \beta_n A' \\ A' & \rightarrow & \alpha_1 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon \end{array}$$

ALGORITMUS 2: Eliminácia ľavej rekurzie

usporiadaj neterminály v nejakom poradí $A_1, \ldots, A_n$
for (každé i od 1 po n)
    for (každé j od 1 po $i-1$) {
        nahraď každé pravidlo tvaru $A_i \rightarrow A_j\gamma$ pravidlami
        $A_i \rightarrow \delta_1\gamma \mid \ldots \mid \delta_k\gamma$, kde
        $A_j \rightarrow \delta_1 \mid \ldots \mid \delta_k$ sú všetky aktuálne A_j-pravidlá
    }
    eliminuj priamu ľavú rekurziu medzi A_i-pravidlami
}

Úpravy gramatiky

3. *Ľavá faktorizácia* ($\rightarrow$ parsovanie zhora-nadol)
   - Využíva sa pri prediktívnom parsovaní
   - $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$ nahradíme:

$$\begin{array}{lcl} A & \rightarrow & \alpha A' \\ A' & \rightarrow & \beta_1 \mid \beta_2 \end{array}$$

Konštrukcie mimo CF jazykov

- Nie celú syntax programovacích jazykov je možné popísať CF gramatikou
- Typické problematické konštrukcie:
  1. Deklarovanie a následné použitie identifikátora $w \rightarrow \mathbf{wcw}$
  2. Volanie procedúr s n argumentami $\rightarrow \mathbf{a^n b^n c^n}$

Parsovanie zhora-nadol

Strom odvodenia sa konštruuje zhora nadol = konštruuje sa ľavé krajné odvodenie

Rekurzívny zostup

- Všeobecná metóda parsovania zhora-nadol
- Môže zahŕňať aj backtracking
  - Keď máme viac pravidiel na výber, skúsime prvé a ak neskôr zistíme nekonzistentnosť vrátime sa a skúšame ďalšie
- Ľavorekurzívne gramatiky môžu viesť k nekonečnému cykleniu

PROCEDÚRA PRE NETERMINÁL A

```
void(A) {
    vyber A-pravidlo A → X1 X2 ... Xk
    for (i = 0 to k) {
        if (Xi je neterminál)
            zavolaj procedúru Xi()
        else if (Xi sa rovná aktuálnemu vstupnému symbolu a)
            posuň sa na vstupe na ďalší symbol;
        else /* nastala chyba */;
    }
}
```

Prediktívne parsovanie

- Špeciálny prípad rekurzívneho zostupu, nie je potrebný backtracking
- Na základe *lookaheadu* a neterminálu *A* sa musíme vedieť jednoznačne rozhodnúť, ktoré *A*-pravidlo použiť
  - Lookahead zvyčajne obmedzený na jeden vstupný symbol
- Vizualizácia: *Prechodové diagramy*
- Implementácia:
  1. Rekurzívne procedúry
  2. Nerekurzívne pomocou zásobníka
- LL(1) gramatiky - definícia neskôr

Prechodové diagramy

Konštrukcia z gramatiky

1. Eliminujeme ľavú rekurziu a vykonáme ľavú faktorizáciu gramatiky
2. Pre každý neterminál A vytvoríme samostatný diagram:
   a) Vytvoríme počiatočný a koncový stav
   b) Pre každé pravidlo $A \to X_1 X_2 \ldots X_n$ vytvoríme cestu z počiatočného do koncového stavu, ktorej šípky sú označené $X_1, \ldots, X_n$.
   Ak $A \to \varepsilon$, potom je šípka označená ε.

- Prechodové diagramy je možné zjednodušiť, ale postupnosti znakov označujúce jednotlivé cesty musia ostať uchované

Prechodové diagramy

Význam prechodov

- Prediktívne parsovanie je možné, iba ak máme deterministické diagramy

1. Prechod na terminál *a*
   - Posun na vstupe o jeden znak doprava (prečítaný symbol musí byť *a*)
2. Prechod na neterminál *A*
   - Pri implementácií pomocou rekurzívnych procedúr: volanie procedúry pre *A*
   - Pri implementácií pomocou zásobníka: pridanie stavu *t* do zásobníka, ak existuje z neho prechod na *A* a vybratie *t* zo zásobníka, ak dosiahneme koncový stav diagramu pre *A*

Pomocné funkcie FIRST a FOLLOW

- Na implementáciu prediktívneho parsera potrebujeme dve pomocné funkcie:
  1. $FIRST(\alpha)$, $\alpha \in (N \cup T)^*$
     - Možina terminálov, ktorými môže začínať reťazec odvodený z α,
     - Ak $\alpha \Rightarrow^* \varepsilon$, potom aj ε patrí do $FIRST(\alpha)$
  2. $FOLLOW(A)$, $A \in N$
     - Množina terminálov a, ktoré sa môžu vyskytnúť hneď vedľa A v nejakej vetnej forme, t.j. existuje odvodenie $S \Rightarrow^* \alpha A a \beta$ pre nejaké α a β
     - Ak A môže byť najpravejší symbol vo vetnej forme, potom aj \$ patrí do $FOLLOW(A)$

Výpočet $FIRST(\alpha)$

- Výpočet $FIRST(X)$ pre všetky symboly gramatiky X
  1. Ak X je terminál, $FIRST(X) = \{X\}$.
  2. Ak X je neterminál a $X \to \varepsilon$ je pravidlo, pridaj ε do $FIRST(X)$
  3. Ak X je neterminál a $X \to Y_1 \ldots Y_k$ je pravidlo,
     - Pridaj a do $FIRST(X)$, ak $a \in FIRST(Y_i)$ a ε je vo $FIRST(Y_1), \ldots, FIRST(Y_{i-1})$
     - Pridaj ε do $FIRST(X)$, ak ε je vo $FIRST(Y_1), \ldots, FIRST(Y_k)$

- Výpočet $FIRST(X_1 \ldots X_n)$
  1. Pridaj $FIRST(X_i) - \{\varepsilon\}$, ak ε je vo $FIRST(X_1), \ldots, FIRST(X_{i-1})$
  2. Pridaj ε, ak ε je vo $FIRST(X_1), \ldots, FIRST(X_n)$

Výpočet $FOLLOW(A)$

1. Pridaj \$ do $FOLLOW(S)$
2. Ak existuje pravidlo $A \to \alpha B \beta$, potom pridaj $FIRST(\beta) - \{\varepsilon\}$ do $FOLLOW(B)$
3. Ak existuje pravidlo $A \to \alpha B$ alebo $A \to \alpha B \beta$, kde $\varepsilon \in FIRST(\beta)$, potom pridaj všetko z $FOLLOW(A)$ do $FOLLOW(B)$

Implementácia - rekurzívne procedúry

- Jedna rekurzívna procedúra pre každý neterminál
- Využíva sa funkcia *FIRST*

1. Rozhodne sa, ktoré pravidlo použiť podľa symbolu na vstupe
   - Vyberie sa pravidlo s pravou stranou α, ak sa čítaný symbol nachádza vo *FIRST*(α)
   - Vyberie sa pravidlo s pravou stranou ε, ak sa čítaný symbol nenachádza vo *FIRST*(α) pre žiadnu pravú stranu
2. Použije vybrané pravidlo "simulovaním"jeho pravej strany
   - Pre každý neterminál sa rekurzívne volá príslušná procedúra
   - Pre každý terminál (token) sa skontroluje, či sa zhoduje s aktuálnym symbolom na vstupe a posunie sa na vstupe o jeden znak doprava

- Na začiatku voláme procedúru pre počiatočný neterminál

Implementácia - rekurzívne procedúry

Množina pravidiel:

$$
\begin{array}{lcl}
S & \rightarrow & \textbf{expr}; \\
 & | & \textbf{if} \ (\ \textbf{expr}\ )\ S \\
 & | & \textbf{for}\ (\ O\,;\ O\,;\ O\ )\ S \\
 & | & \textbf{other}
\end{array}
\qquad
\begin{array}{lcl}
O & \rightarrow & \varepsilon \\
 & | & \textbf{expr}
\end{array}
$$

PROCEDÚRY PRE NETERMINÁLY *S*, *O* A "MATCHOVACIA" PROCEDÚRA

```
void S() {
    switch (lookahead) {
    case expr:
        match(expr); match(';'); break;
    case if:
        match(if); match('('); match(expr);
        match(')'); S(); break;
    case for:
        match(for); match('(');
        O(); match(';'); O(); match(';'); O();
        match(')'); S();  break;
    case other:
        match(other);  break;
    default:
        report("syntax error");
    }
}
```

```
void O() {
    if (lookahead == expr)
        match(expr);
}

void match(terminal t) {
    if (lookahead == t)
        lookahead = nextTerminal;
    else
        report("syntax error");
}
```

Implementácia - zásobník

- Nerekurzívna metóda
- Parser používa zásobník na uloženie nespracovanej postupnosti symbolov gramatiky, na začiatku je tam poč. neterminál
- Pravidlo použiteľné pre aktuálny neterminál a vstupný symbol sa hľadá v *parsovacej tabuľke*
  - Dvojrozmerné pole $M[A, a]$, kde A je neterminál a a terminál alebo endmarker \$
  - Pri konštrukcii sa využívajú obe funkcie *FIRST* a *FOLLOW*

Implementácia - zásobník

Konštrukcia parsovacej tabuľky

Vstup: Gramatika G.
Výstup: Parsovacia tabuľka M.

1. Pre každé pravidlo $A \to \alpha$ vykonaj kroky 2 a 3.
2. Pre každý terminál $a \in \mathit{FIRST}(\alpha)$ pridaj $A \to \alpha$ do $M[A, a]$.
3. Ak $\varepsilon \in \mathit{FIRST}(\alpha)$, pridaj $A \to \alpha$ do $M[A, b]$ pre každý terminál $b \in \mathit{FOLLOW}(A)$. Ak $\varepsilon \in \mathit{FIRST}(\alpha)$ a $\$ \in \mathit{FOLLOW}(A)$, pridaj $A \to \alpha$ do $M[A, \$]$.
4. Každé nedefinovaný záznam označ ako chybu.

Implementácia - zásobník

Algoritmus parsovania

priraď do *ip* smerník na prvý symbol w;
priraď do X vrchný symbol zásobníka;
while $(X \neq \$)$ { /* zásobník nie je prázdny */
    if $(X = a)$
        vyber X zo zásobníka a posuň sa s *ip* na ďalší symbol w;
    else if (X je terminál)
        error();
    else if ($M[X, a]$ je chybové políčko)
        error();
    else if $(M[X, a] = X \rightarrow Y_1 Y_2 \ldots Y_k)$ {
        daj na výstup pravidlo $X \rightarrow Y_1 Y_2 \ldots Y_k$;
        vyber X zo zásobníka;
        vlož $Y_k, Y_{k-1} \ldots Y_1$ do zásobníka, tak, že Y_1 je na vrchu;
    }
    priraď do X vrchný symbol zásobníka;
}

Zotavenie z chýb

- Dva druhy chýb:
  1. Terminál na vrchu zásobníka nie je zhodný s terminálom na vstupe
  2. Na vrchu zásobníka je neterminál A, ďalším vstupným symbolom je a a záznam v parsovacej tabuľke $M[A, a]$ je prázdny
- Zotavenie v móde paniky:
  - Synchronizačné tokeny sú napr. všetky symboly z FOLLOW(A) pre neterminál A
- Zotavenie na úrovni frázy:
  - Prázdne záznamy v parsovacej tabuľke sa vyplnia smerníkmi na procedúry ošetrujúce danú konkrétnu chybu

LL(1) gramatiky

- LL(1) je trieda CF gramatík, pre ktorú vieme zostrojiť prediktívne parsery
  - L - vstup sa číta zľava doprava
  - L - konštruuje sa ľavé krajné dovodenie
  - (1) - výhľad dopredu je obmedzený na 1 symbol
- Nejednoznačné a ľavorekurzívne gramatiky nie sú LL(1)
  - Existujú ale gramatiky, ktoré ani po úpravách nie sú LL(1)
- Gramatika je LL(1) práve vtedy, ak pre každú dvojicu rôznych pravidiel $A \to \alpha \mid \beta$ platí
  1. $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$.
  2. Ak $\varepsilon \in FIRST(\beta)$, potom $FIRST(\alpha) \cap FOLLOW(A) = \emptyset$.
     Ak $\varepsilon \in FIRST(\alpha)$, potom $FIRST(\beta) \cap FOLLOW(A) = \emptyset$.

  $\to$ vieme zostrojiť parsovaciu tabuľku

Výhody a nevýhody

(+) Parsery zhora-nadol sa ľahšie ručne implementujú ako parsery zdola-nahor

(−) Gramatika je po úpravách neprehľadná a ťažšie sa pre ňu definuje preklad